CptS 360 (System Programming) Unit 16: Network IPC (Sockets)

Bob Lewis

School of Engineering and Applied Sciences
Washington State University

Spring, 2022

Motivation

- Processes need to talk to each other across a network.
- Lots of servers depend on networked IPC.
- Sockets are the basis of all Internet applications.
- Network intrusions happen over sockets.

References

- Stevens & Rago Ch. 16
- *man* pages
- Stones & Matthew "Beginning Linux Programming"

Introduction

- So far, our IPC has been within a single system.
- This reflects POSIX's mainframe origins.
- How about between different computers?
- Network transfers "packets" of data between computers.
- Packets contain
    - format
    - routing information
    - data
- 4.4BSD devised "sockets"
    - "open a socket" is now part of the culture (used in techy TV shows)

socket(2)

- creates a socket
- returns a file descriptor, just like *open(2)*
- args specify
  - domain
    kind of communication, esp. address format
  - type
    of connection
  - protocol
    when domain and type map to multiple protocols (usually the default is what you want, so set to 0)
- This call does *not* make the socket visible outside of your process. (That comes later.)

Socket Domains

- `AF_INET`
  IPv4 – (allows 32-bit address) the Internet as we know it
- `AF_INET6`
  IPv6 – (allows 128-bit address) the Internet as we would eventually like to know it one of these days eventually, maybe (Standardized in 1998, it's the default on many mobile phone systems and it's *slowly* becoming reality.)
  - Are you using it? `https://test-ipv6.com`
  - How many places are using it? `https://www.google.com/intl/en/ipv6`
- `AF_UNIX` (aka `AF_LOCAL`)
  intramachine (better alternative to XSI messsages)
- `AF_UNSPEC`
  "any" domain (not sure how this works)

Socket Types

- `SOCK_DGRAM`
  (“datagram”) fixed length, connectionless, unreliable
- `SOCK_STREAM`
  sequenced, reliable, bidirectional, “connection oriented”
- `SOCK_SEQPACKET`
  fixed-length, sequenced, reliable, “connection oriented”
- `SOCK_RAW`
  datagram interface to IP

Byte Ordering

- big- vs. little-endian
  remember mnemonic for which is which.
- *byteorder(3)*

Address Formats

- determined by domain
- generic sockaddr `struct`:

  ```
  struct sockaddr {
    sa_family_t  sa_family;
    char sa_data[];
    ...
  }
  ```

  (another approach to polymorphism)
- (see `/usr/include/netinet/in.h`)

Address Lookup: Host Name

- *gethostent(3)*
  What hosts are known? (A small subset, as it does not take nameservers into account.) This works just like *getpwent(3)*: Call repeatedly to get successive entries until it returns `NULL`.
- *gethostbyaddr(3)*
  obsolete (see below)
- *gethostbyname(3)*
  obsolete (see below)

(run `demos/d`n`_get_network_stuff/list_hosts`)

Address Lookup: Network, Protocol, Services

These provide information about the local host. Like *gethostent(3)*, they work like *getpwent(2)*.

- *getnetent(3)*, *getnetbyaddr(3)*, etc.
  What networks are available? Usually, it's just the Internet.
  (run `demos/d`n`_get_network_stuff/list_networks`)
- *getprotoent(3)*, *getprotobyname(3)*, etc.
  What protocols are available? Usually lots, but IP and TCP are the most often used.
  (run `demos/d`n`_get_network_stuff/list_protocols`)
- *getservent(3)*, *getservbyname(3)*, etc.
  What services are being provided? Usually lots.
  (run `demos/d`n`_get_network_stuff/list_services`)

Finding Remote Host Information

- *getaddrinfo(3)*
  - replaces *gethostbyname(3)* and *gethostbyaddr(3)*, which are considered obsolete

(run `demos/d`n`_describe_node`)

Port Numbers

A small integer indicates the service associated with an address. Certain ranges are designated:

terminology	range
"well-known", "system" (root only)	0 - 1023
"registered" (by the IANA[1])	1024 - 49151
"dynamic", "private", "ephemeral" (Have at 'em!)	49252 - 65535

(See `https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt` for official or `/etc/services` or call *getservent(3)* and friends for platform-specific assignments.)

[1]Internet Assigned Numbers Authority

Associating Addresses with Sockets

- *bind(2)*
- *getsockname(2)*
  inverse of *bind(2)*
- *getpeername(2)*
  peer is socket address at the other end of the socket

Connection Establishment

- If we're dealing with a connection-oriented network service (Q: such as ?)...
- *connect(2)*

(see `demos/d`n`_include/connect_retry.c`)

Data Transfer

If there's a connection and all we want to do is send data, we can use good old

- *read(2)*
- *write(2)*

(or even stdio), but to handle

- connectionless transfer
- multiple clients
- send out-of-band data
- other options

we need ...

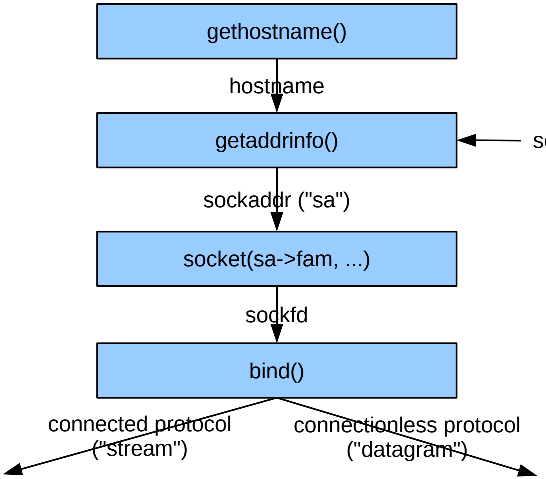
Data Transfer: Sending

- *send(2)*
  - connected only
  - note flags argument
- *sendto(2)*
  - works with connectionless transfer
- *sendmsg(2)*
  - send from multiple buffers (like *writev(2)*)
  - can also be used to send control (aka "ancillary" or "out of band") information:
    - packet interface information
    - credentials (PID, PPID, PGID, SID, UID, SID, etc.)
    - set of file descriptors (which could convey permissions)

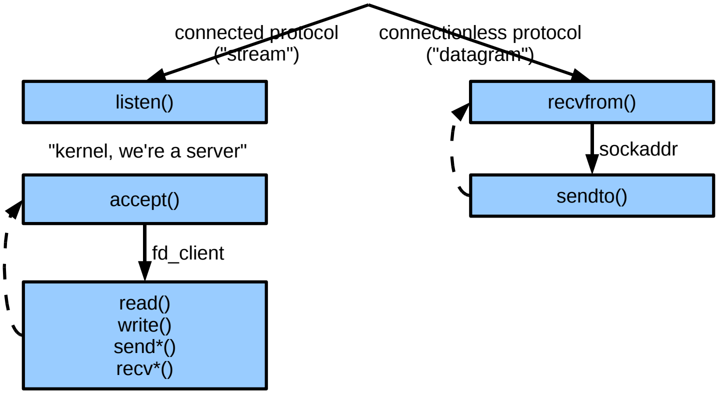
Data Transfer: Receiving

- *recv(2)*
  - connected only
- *recvfrom(2)*
  - works with connectionless transfer
- *recvmsg(2)*
  - receive into from multiple buffers (like *readv(2)*)
  - can also be used to receive control information

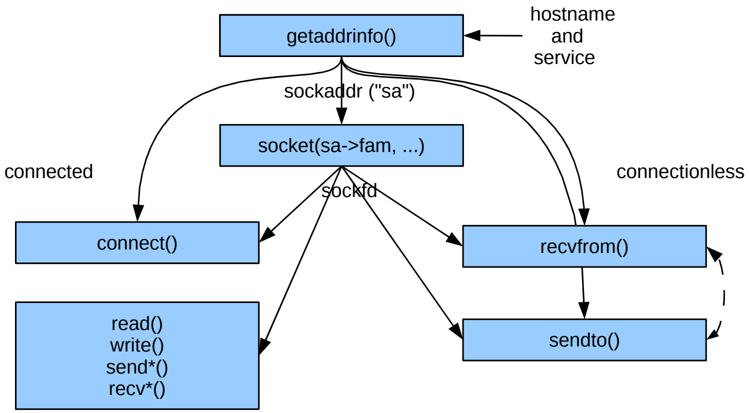
Server Chronology I



Server Chronology II



Client Chronology



Internet Client/Server Demos

The following demos are available:

- (see `demos/dn_sr_client_server_datagram`)
  This is (more or less) Stevens & Rago's original datagram server demo. It is not maintained.
- (see `demos/dn_sr_client_server_stream`)
  This is (more or less) Stevens & Rago's original stream server demo. It is not maintained.
- (run `demos/dn_ruptime`)
  This is a refactored version of Stevens & Rago's original datagram server demo. It is maintained on WSUTC systems.

Out-of-Band Data

- sent ahead of any queued data
- TCP allows one byte
- UDP doesn't allow
- recipent gets `SIGURG`

Asynchronous vs. Nonblocking I/O

- asynchronous
  - start now
  - return immediately
  - I'll check later.
- non-blocking
  - start now
  - return immediately *if I/O can't be completed*
  - I'm not going to bother checking.